UNIL | Université de Lausanne

DEEP EQUILIBRIUM NETS

Tuesday, March 25th, 2025

Simon Scheidegger (UNIL)

https://onlinelibrary.wiley.com/doi/epdf/10.1111/iere.12575

Replication codes here: https://github.com/sischei/DeepEquilibriumNets

Abstract Problem Formulation

i) Contemporary dynamic models: heterogeneous & high-dimensional

ii) Want to compute global solutions to dynamic stochastic models with high-dimensional state spaces

→ Have to **approximate** and **interpolate** high-dimensional functions on irregular-shaped geometries

→ Problem: curse of dimensionality

iii) **Want to alleviate the curse of dimensionality**

iv) **Want locality of approximation scheme**

v) **Speed-up** → potentially access contemporary HPC systems

Our solution: Deep Equilibrium Nets

→ Solving, e.g., rich OLG models numerically is a **formidable task**.

→ Models are often formulated in a **stylized** fashion to remain computationally **tractable**.

→ We develop a **generic solution framework** based on **neural networks** to solve highly-complex dynamic stochastic models.

Key ideas:

1. Use the **the implied error in the optimality conditions**, as **loss function**.
2. Learn the equilibrium functions with stochastic gradient descent.
3. Take the (training) data points from a simulated path
   → can be generated at **virtual zero cost**.

How to find good parameters ρ?

The standard way:

Step 1: get "labelled data" $\mathcal{D} := \left\{ (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_{|\mathcal{D}|}, \mathbf{y}_{|\mathcal{D}|}) \right\}$ where $\mathbf{y}_i = \mathbf{f}(\mathbf{x}_i)$ is the correct output → **Supervised learning**

Step 2: Define a loss function, for example:

$$l_\rho := \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}} \left(\mathbf{y}_i - \mathcal{N}_\rho (\mathbf{x}_i) \right)^2$$

Step 3: Adjust the parameters to minimize the loss via (**stochastic**) **gradient descent**:

$$\rho_i^{\text{new}} = \rho_i^{\text{old}} - \alpha^{\text{step}} \frac{\partial l_{\rho^{\text{old}}}}{\partial \rho_i^{\text{old}}}$$

the step-width α^{step} is called the "**learning rate**" and the process of adjusting the parameters is called "learning".

How to find good parameters ρ?

- The deeper and larger the neural net becomes, the more flexible it is as a function approximator, …

  ... but the more data it will need

  → rule of thumb: **#observations/parameters** ~ 10x (Marsland (2014).

- In economic applications, the standard way of solving for equilibria is to use time iteration → **“small” data**.
  (e.g., Sparse Grids (Brumm & Scheidegger (2017), Krüger & Kübler (2004), Judd et al. (2014))

The issue with time iteration

- **Time Iteration – collocation** (see, e.g., Judd (1998), and references therein)

```
1. Select a grid $G$, and a policy function $f^{\text{start}}$. Set $f^{\text{next}} \equiv f^{\text{start}}$.
2. Make one time iteration step:
       1. For all $g \in G$, find $f(g)$ that solves
          the Period-to-Period Equilibrium Problem given $f^{\text{next}}$.
       2. Use solutions at all grid points $G$ to interpolate
          $f$( how?).
3. Check error criterion: If $\|f - f^{\text{next}}\|_\infty < \epsilon$, report solution: $\tilde{f} = f$
   Else set $f^{\text{next}} \equiv f$ and go to step 2.
```

→ Solving non-linear sets of equations in every iteration can be a daunting task.

→ What if non-linear solver does not converge? Construction of approximator may crash.

→ Can the dynamic economic problem at hand be mapped onto a grid?

An “economic” loss function

- **Novelty**: we propose an “economic” loss function:

$$l_\rho := \frac{1}{N_{\text{path length}}} \sum_{\text{x}_\text{i} \text{ on sim. path}} \left(\mathbf{G}\left(\mathbf{x}_i, \mathcal{N}_\rho\left(\mathbf{x}_i\right)\right)\right)^2$$

  where we use $\mathcal{N}_\rho$ to simulate a path.

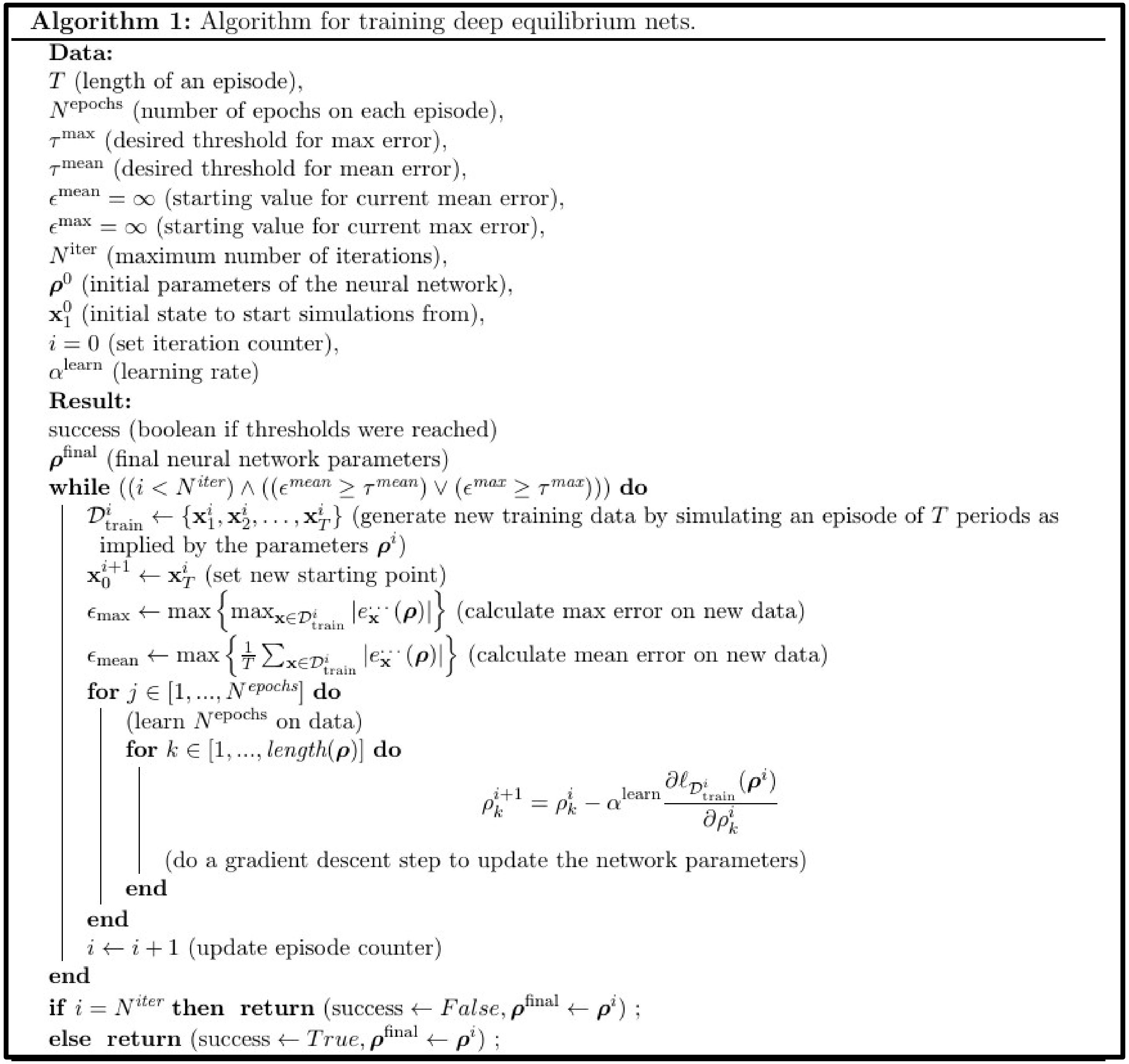
- **G** is chosen such that the **true equilibrium policy f(x)** is defined by

  G(**x**, **f**(**x**)) = 0 ∀**x**.

- **G(.,.): implied error in the optimality conditions** (unit-free Euler errors)

- Therefore, there is **no need for labels** to evaluate our loss function.
  → **Unsupervised Machine Learning**.

Training Deep Equilibrium Nets

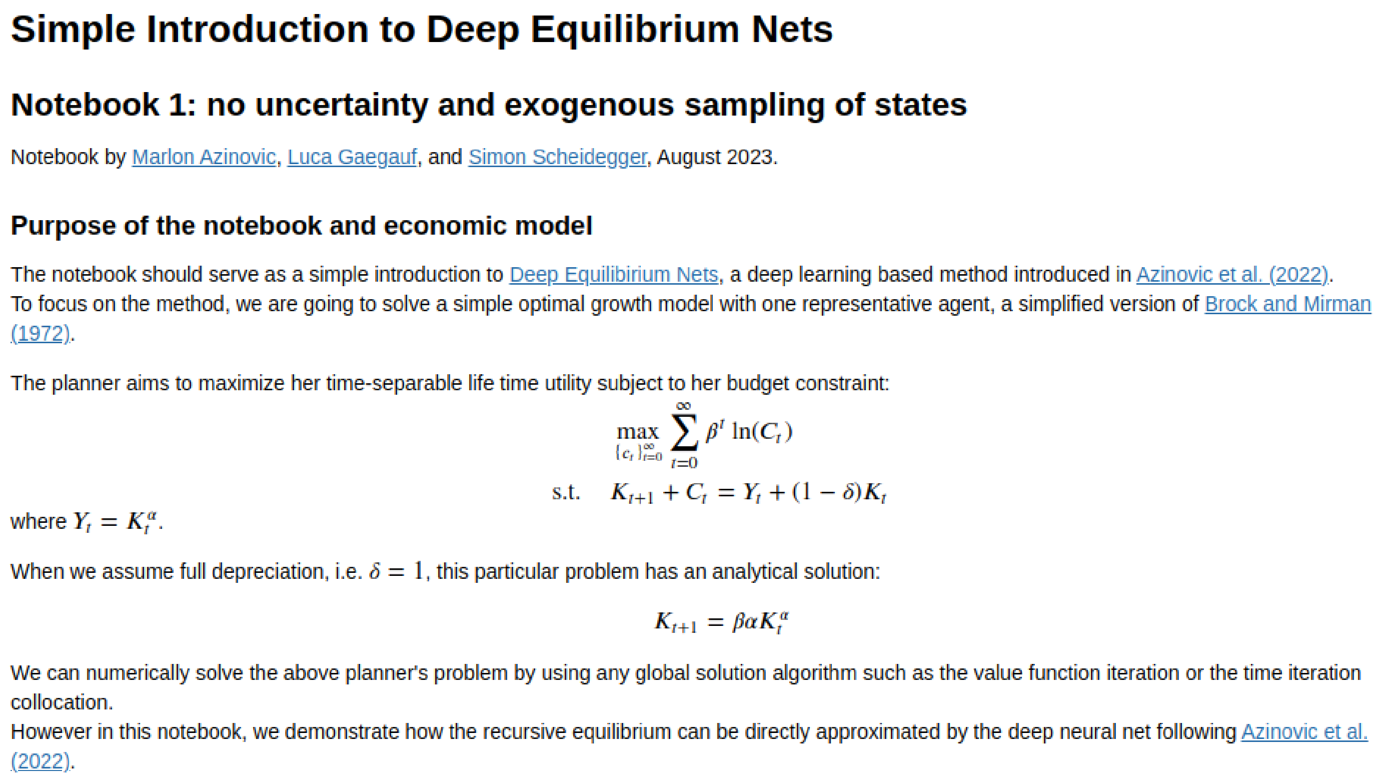


Algorithm 1: Algorithm for training deep equilibrium nets.

```
Data:
$T$ (length of an episode),
$N^{\text{epochs}}$ (number of epochs on each episode),
$\tau^{\max}$ (desired threshold for max error),
$\tau^{\text{mean}}$ (desired threshold for mean error),
$\epsilon^{\text{mean}} = \infty$ (starting value for current mean error),
$\epsilon^{\max} = \infty$ (starting value for current max error),
$N^{\text{iter}}$ (maximum number of iterations),
$\boldsymbol{\rho}^0$ (initial parameters of the neural network),
$\mathbf{x}_1^0$ (initial state to start simulations from),
$i = 0$ (set iteration counter),
$\alpha^{\text{learn}}$ (learning rate)
Result:
success (boolean if thresholds were reached)
$\boldsymbol{\rho}^{\text{final}}$ (final neural network parameters)
while $((i < N^{iter}) \wedge ((\epsilon^{mean} \geq \tau^{mean}) \vee (\epsilon^{max} \geq \tau^{max})))$ do
    $\mathcal{D}_{\text{train}}^i \leftarrow \{\mathbf{x}_1^i, \mathbf{x}_2^i, \ldots, \mathbf{x}_T^i\}$ (generate new training data by simulating an episode of $T$ periods as
     implied by the parameters $\boldsymbol{\rho}^i$)
    $\mathbf{x}_0^{i+1} \leftarrow \mathbf{x}_T^i$ (set new starting point)
    $\epsilon_{\max} \leftarrow \max\left\{\max_{\mathbf{x}\in\mathcal{D}_{\text{train}}^i} |e_{\mathbf{x}}^{\cdots}(\boldsymbol{\rho})|\right\}$ (calculate max error on new data)
    $\epsilon_{\text{mean}} \leftarrow \max\left\{\frac{1}{T}\sum_{\mathbf{x}\in\mathcal{D}_{\text{train}}^i} |e_{\mathbf{x}}^{\cdots}(\boldsymbol{\rho})|\right\}$ (calculate mean error on new data)
    for $j \in [1, ..., N^{epochs}]$ do
        (learn $N^{\text{epochs}}$ on data)
        for $k \in [1, ..., length(\boldsymbol{\rho})]$ do
            $$\rho_k^{i+1} = \rho_k^i - \alpha^{\text{learn}} \frac{\partial \ell_{\mathcal{D}_{\text{train}}^i}(\boldsymbol{\rho}^i)}{\partial \rho_k^i}$$
            (do a gradient descent step to update the network parameters)
        end
    end
    $i \leftarrow i + 1$ (update episode counter)
end
if $i = N^{iter}$ then return (success $\leftarrow False, \boldsymbol{\rho}^{\text{final}} \leftarrow \boldsymbol{\rho}^i$) ;
else return (success $\leftarrow True, \boldsymbol{\rho}^{\text{final}} \leftarrow \boldsymbol{\rho}^i$) ;
```

2. II. A simple benchmark model

- Lets check-out the notebook:
  day2/code/01_Brook_Mirman_1972_DEQN.ipynb



Simple Introduction to Deep Equilibrium Nets

Notebook 1: no uncertainty and exogenous sampling of states

Notebook by Marlon Azinovic, Luca Gaegauf, and Simon Scheidegger, August 2023.

Purpose of the notebook and economic model

The notebook should serve as a simple introduction to Deep Equilibirium Nets, a deep learning based method introduced in Azinovic et al. (2022).
To focus on the method, we are going to solve a simple optimal growth model with one representative agent, a simplified version of Brock and Mirman (1972).

The planner aims to maximize her time-separable life time utility subject to her budget constraint:

$$\max_{\{c_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t \ln(C_t)$$
$$\text{s.t.} \quad K_{t+1} + C_t = Y_t + (1-\delta)K_t$$

where $Y_t = K_t^{\alpha}$.

When we assume full depreciation, i.e. $\delta = 1$, this particular problem has an analytical solution:

$$K_{t+1} = \beta\alpha K_t^{\alpha}$$

We can numerically solve the above planner's problem by using any global solution algorithm such as the value function iteration or the time iteration collocation.
However in this notebook, we demonstrate how the recursive equilibrium can be directly approximated by the deep neural net following Azinovic et al. (2022).